Carolina Bolnykh

Netflix Analysis: IMDB Ratings

Data Subsetting: Movies vs tv-shows

For my part of the group project, I will be using decision tree models for these datasets to allow me to see which features are deemed more important in predicting the imdb rating category. The way that the decision tree models make decisions is by splitting on entropy or GINI index, both of which have a sooner split indicating a higher significance on that factor. As I completed my preliminary data analysis, I noticed that there were 2 main groups of data points: Movies and Shows. Since movies and shows are different in nature, I have decided to split the dataset into 2 separate data frames: **finalcolsMovie** (which includes only movie entries from our final merged dataset), and **finalcolsShows** (which includes only tv-show entries from our final merged dataset). In part 2 of the project, we hypothesized that movies and tv-shows will have different factors that go into predicting their IMDB scores, which is why this is a crucial step towards building our model. Doing this will also give us the opportunity to do feature engineering and work with features what tv-shows have (like #of seasons), that movies do not.

```
71  # Carolina's Part Begins:
72  finalcols=merge(titles,pmactors,by="id",all.x = TRUE)
73  # Split into 2 separate dataframes to analyze movies and TV shows separately
74  finalcolsMovie <- finalcols %>% filter(type == "MOVIE")
75  finalcolsMovie <- finalcolsMovie[!is.na(finalcolsMovie$imdb_score), ]
76  finalcolsShow <- finalcols %>% filter(type == "SHOW")
77  finalcolsShow <- finalcolsShow[!is.na(finalcolsShow$imdb_score), ]
78
```

Categorizing IMDB scores: transform numerical to categorical

An important part in building decision trees is defining the ”target” column, which you want your model to predict. Since I am working with classification trees, they are going to make categorical predictions. Our final merged datasets contain IMDB scores for movies and tv-shows, but they are numerical. Out of all the records, the highest value is a 9.6 and lowest value is a 1.5, so it wouldn’t make sense to split from 0 to 10 equally.To figure out how to split the data best, I generated a 5-number summary for the dataset (min, Q1, median, Q3, max), and split the data based on those results. This transformed the original dataset’s target IMDB values into “poor”, “average”, “good”, and “excellent” in a way that was proportional to the distribution.

```
93  # obtain a 5-number summary for movie data to gain more insights into how to categorize the imbd lables
94  summary(finalcolsMovie$imdb_score)
95
96  # split: (poor: 1.5-5.6, 5.6-6.4: average, 6.4-7.1: good, 7.1-9.1: excellent) based on our 5-number summary
97  categorySplit <- c(1.5, 5.6, 6.4, 7.1, 9.1)
98  categories <- c("Poor", "Average", "Good", "Excellent")
99  finalcolsMovie$imdbCategory <- cut(finalcolsMovie$imdb_score, breaks = categorySplit, labels = categories, include.lowest = TRUE)
```

Building Classification Trees: Movies

I chose a 67% / 33% split for splitting our data into testing and training, and created labels that can be used in testing our accuracy

```
108 set.seed(12345)
109 indMovie <- sample(2, nrow(finalcolsMovie), replace=TRUE, prob=c(0.67, 0.33))
110
111 # training & testing labels for movies
112 movies_training <- finalcolsMovie[indMovie==1, c(1:20)]
113 movies_test <- finalcolsMovie[indMovie==2, c(1:19)]
114 movies_trainLabels <- finalcolsMovie[indMovie==1, 20]
115 movies_testLabels <- finalcolsMovie[indMovie==2, 20]
116
```

Decision Tree #1: Analyzing all Attributes

The first decision tree I decided to build was one that takes into consideration all of the original attributes in the dataset to use as a baseline for further feature extraction:

```
# Decision Tree
library(rpart)
library(DMwR)
library(rpart.plot)
# first decision tree with all attributes
ctree <- rpart(imdbCategory ~
               release_year + age_certification + runtime
               + genres + production_continent,
               data=movies_training,method="class")
library("rpart.plot")
```

This decision tree produced an accuracy of 0.336175, which I obtained through the following command:

```
# Tree 1: Using all features (.33 accuracy)
predictions <- predict(ctree, movies_test, type = "class")
accuracy <- sum(predictions == movies_testLabels) / length(movies_testLabels)
accuracy

```

Decision Tree #2: Popularity Attributes

Since one my original hypotheses was that actor popularity and director popularity have a high influence in a movie's IMDB score, I wanted to build a model that only takes into consideration the mean actor popularity and director popularity (both obtained through merging with the oscars dataset) in order to classify the IMDB score class.

```
# Tree 2: By popularity (directors actors)
## Popularity: getting about the same accuracy as with all (and we cut down the # of features drastically): .28
ctree2 <- rpart(imdbCategory ~  directormedian + actormedian, data=movies_training, method="class")
predictions2 <- predict(ctree2, movies_test, type = "class")
accuracy2 <- sum(predictions2 == movies_testLabels) / length(movies_testLabels)
accuracy2

```

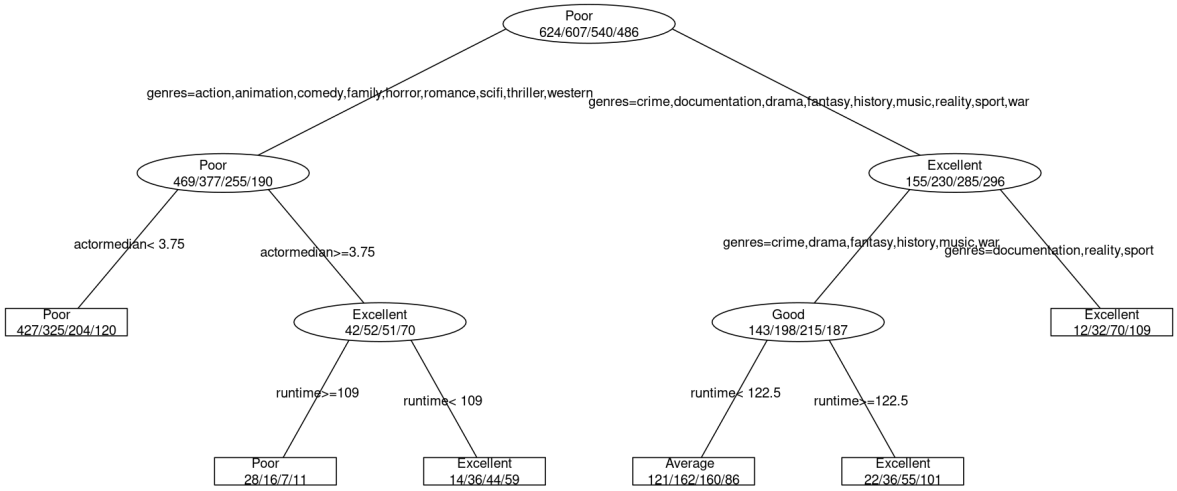
The accuracy for this model is 0.2842, which is close to the accuracy that we got by using all of the features. Through this, we can deduce that just using 2 predictors (actor popularity, director popularity) produces almost the same accuracy as using all of the predictor variables in our original dataset. Since just those 2 together provide almost 30% accuracy, they can be deemed as important factors in IMDB prediction. Next, I want to see if we will be able to get a more powerful prediction by combining the merged features with the original dataset's features:

Decision Tree #3: Combination Model

```
# Tree 3: Combining all with popularity (directors, actors)
## Popularity: getting about the same accuracy as with all (and we cut down the # of features drastically):
ctreeCombined <- rpart(imdbCategory ~  release_year + age_certification + runtime
                + genres + production_continent + directormedian + actormedian,
                data=movies_training, method="class")
predictionsCombined <- predict(ctreeCombined, movies_test, type = "class")
accuracyCombined <- sum(predictionsCombined == movies_testLabels) / length(movies_testLabels)
accuracyCombined
```

This model gives us an accuracy of **0.3507**, which shows us that including actor and director popularity information gives us a 2% improvement in accuracy from model 1, and 7% improvement from model 2. We get the best performing model when we include all attributes and the new engineered features that indicate actor & director popularity. When analyzing the

splits of this model, we can print the tree in R and get this as a result:



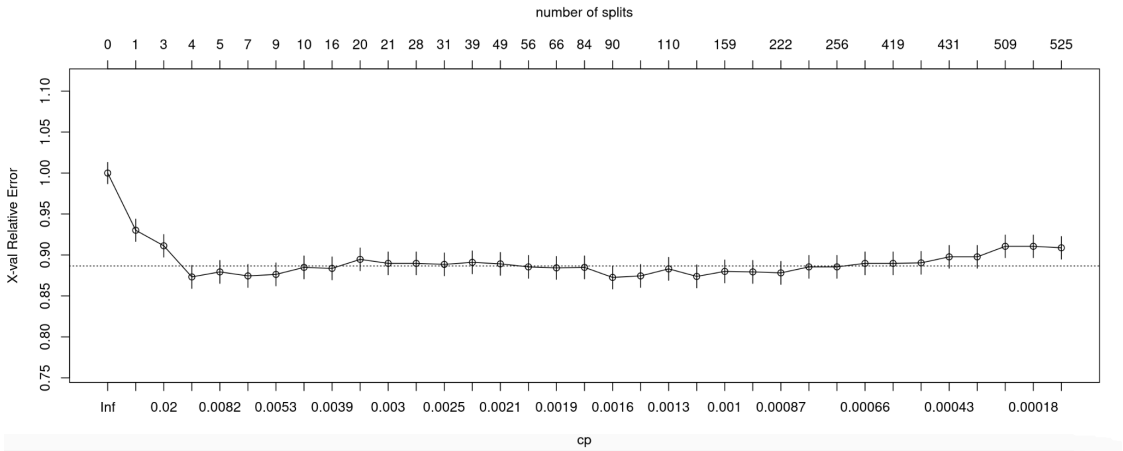
As we can see in the output above, the best performing model has used movie **genre** as the first feature to split on, meaning that it was the split that provided the greatest reduction in impurity. Next, it splits on **actormedian** (which is the popularity attribute), and then **runtime.** These 3 features are the features that give the model the most information in order to classify a movie's imdb rating into "poor", "average", "good", or "excellent".

Movies Model Improvement: Pruning

In an attempt to bring up our accuracy from 35%, I want to use the Minimal Cost Complexity Pruning technique, which entails growing our tree too large and then pruning all branches with CP (complexity parameter) less than a certain threshold. We grow the tree "too large" by setting minsplit = 5 and cp=0.

```
159 # Trying to improve accuracy: "grow the tree too large to prune" using the Combined Decision Tree:
160 ctreeLarge <- rpart(imdbCategory ~
161                       release_year + age_certification + runtime
162                     + genres + production_continent + actormedian + directormedian,
163                     data=movies_training,method="class",
164                     control=rpart.control(minsplit=5,cp=0))
```

I plot CP values in order to figure out the best out to use in pruning: plotcp(ctreeLarge,upper="splits")



A common technique for finding the optimal cp value is to look for the leftmost value for which the error lies below the horizontal line, here it looks to be around 0.0039. Next, we prune all nodes with complexity less than cp=0.0039

```
plotcp(ctreeLarge,upper="splits")
#We prune all nodes with complexity less than cp=0.03
ctree4 <- prune(ctreeLarge,cp=0.0039,)
prettyTree(ctree4)
predictions4 <- predict(ctree4, movies_test, type = "class")
accuracy4 <- sum(predictions4 == movies_testLabels) / length(movies_testLabels)
accuracy4
ctree4
```

Pruning provides us with an *accuracy4* value of 38% on the pruned combination model.

Building Classification Trees: TV-Shows

For building classification trees for tv-shows, I will be using the same approach as movies (building a combination-feature model using the merged dataset and new columns), but including an additional feature: number of seasons (which is 0 for all movie entries in the dataset). Additionally, I want to investigate whether or not the # of seasons feature has an impact on the final prediction. This will be helpful in seeing if it was worth splitting up the dataset into 2 parts, or if it would have been fine just having all the data in one dataset but dropping the seasons column. Repeating the same histogram 5-number summary procedure on tv-show imdb scores, we have found the following split for shows: (poor: 2-6.4, 6.4-7.1: average, 7.1-7.7: good, 7.7-9.6: excellent)

Decision Tree #1: Analyzing all Attributes - No # of Seasons

```
library(rpart)
library(DMwR)
library(rpart.plot)
ctree <- rpart(imdbCategory ~
                release_year + age_certification + runtime
               + production_continent + genres + actormedian,
              data=shows_training,method="class")
library("rpart.plot")
predictions <- predict(ctree, shows_test, type = "class")
accuracy <- sum(predictions == shows_testLabels) / length(shows_testLabels)
accuracy
```

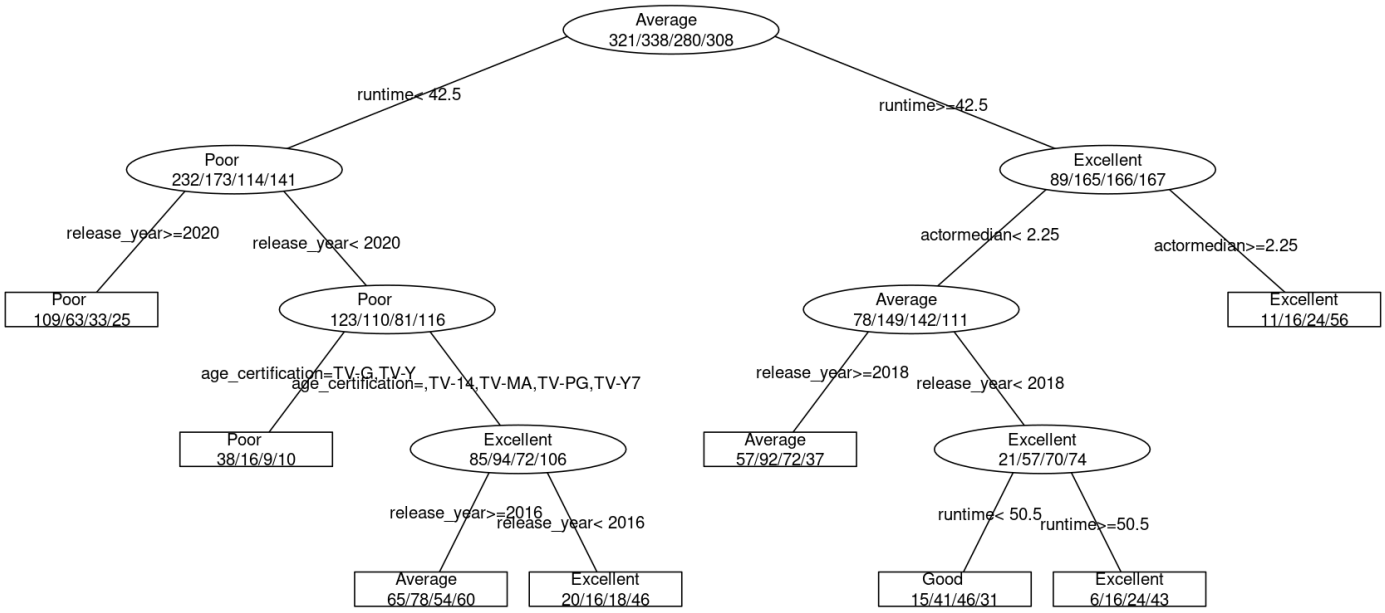
Creating a decision tree with all attributes except seasons gives us an accuracy of 0.3757.

Decision Tree #2: Analyzing all Attributes - Including # of Seasons

```
library(rpart)
library(DMwR)
library(rpart.plot)
ctree <- rpart(imdbCategory ~
                release_year + age_certification + runtime
               + production_continent + genres + actormedian + seasons,
              data=shows_training,method="class")
library("rpart.plot")
predictions <- predict(ctree, shows_test, type = "class")
accuracy <- sum(predictions == shows_testLabels) / length(shows_testLabels)
accuracy
```

Creating a decision tree with all attributes (including actor popularity and number of seasons), which gives us the same accuracy of 0.3757. Therefore, # of seasons is not contributing to the classifying power of our decision tree.

The prettyTree() function in R is ran on Decision Tree #1, and displays the tree below. The results indicate that **runtime, release year, and actor median** are the features that give the model the most information in order to classify a movie's imdb rating into "poor", "average", "good", or "excellent". The first feature that the model split on was runtime, meaning that runtime initially provided the maximum reduction in class impurity.



Summary

Through feature engineering, feature selection, data analysis, and classification trees, we have analyzed how different factors play a role in classifying IMDB scores of movies and tv-shows. My best performing model indicates that the 3 most informative features in reducing impurity for IMDB classification of tv-shows are *runtime, release year, and actor median,* where the first/most informative feature is *runtime*. The identifying features for movies are *genres*, *actor median*, and *runtime*, where the first/most informative feature is *genre*. We have proven our hypothesis that since shows and movies are different in nature, they will have different factors that influence their IMDB score (even though they come from the same dataset with the same features).